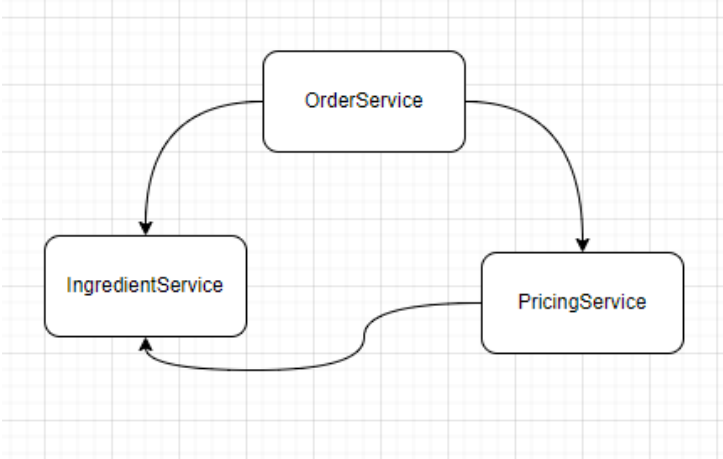
Programação Visual
Trabalho de Laboratório nº 8

Objetivo	MVC – Introdução. Utilização de *ViewComponents* e utilização/definição de serviços
Programa	Pretende-se criar um serviço de encomenda de Pizzas.
Regras	Implementar o código necessário e testar no fim de cada nível. Use as convenções de codificação adotadas para a linguagem C# e para o modelo MVC.

Nível 1

- Obtenha os materiais fornecidos, descompacte e abra a solução.
- Na solução vai encontrar 2 serviços já implementados: `IngredientService` e o `PricingService`, que por sua vez depende do primeiro. Existe também um terceiro serviço `OrderService`, este já conta com uma interface com dois métodos que vamos ter de implementar.
- A arquitetura desta aplicação envolve 3 serviços com dependências entre si.



- Sendo que o OrderService depende de ambos os serviços para o seu funcionamento vamos começar por injetar estes através de **Dependecy Injection**.
- No OrderService, crie dois campos privados apenas de leitura: `_ingredientService` (IIngredientService) e `_princingService` (IPricingService).
- Crie um construtor que recebe dois parâmetros: `ingredientService` (IIngredientService) e `princingService` (IPricingService). No construtor atribua o valor recebido por parâmetro aos seus campos definidos anteriormente.
- Efetue o desenvolvimento do método `GetIngredients`, usufruindo dos serviços que acabou de injetar.

Nota: Repare que este método retorna uma coleção do tipo `OrderIngredient`, que é composto por um `Ingredient` e um preço.

Nota: Os métodos oferecidos pelos serviços injetos retornam um tipo `Task<T>`. Para obter o resultado deve utilizar o `await` ao chamar os métodos. Exemplo:

```
var ingredients = await _ingredientService.GetIngredients();
```

Programação Visual
Trabalho de Laboratório nº 8

Nível 2

- Para implementar o método `OrderPizza` terá de retornar um `OrderResult`, que é composto pelo objeto que foi encomendado (pizza) e pelo preço total da pizza.
- Para calcular o preço total da pizza, terá de somar o preço da base, do tamanho e dos ingredientes. Repare que o serviço PricingService usa entidades diferentes para o tamanho e base de pizza, para resolver esta dependência aplica-se o conceito de ***Mappings***, vai encontrar nos materiais uma classe estática `PizzaMappings` que contém dois métodos de extensão que deverá utilizar para converter os modelos do OrderService para modelos do PricingService.

Nota informativa: Isto é um acontecimento perfeitamente normal quando se trabalha com múltiplos serviços. Uma das ferramentas mais utilizadas é o AutoMapper, que permite configurar estes mapeamentos de forma genérica e aplicá-los quando necessário.

- Realize a implementação do método OrderPizza utilizando os métodos de extensão fornecidos para fazer as chamadas ao PricingService.

Nível 3

- No Program.cs registe os serviços utilizado o código a baixo. Este deve ser colocado antes da linha onde se instancia a variável app.

```
builder.Services.AddSingleton<IIngredientService,
IngredientService>();
builder.Services.AddSingleton<IPricingService, PricingService>();
builder.Services.AddTransient<IOrderService, OrderService>();
```

Nota: Pode ler mais sobre as diferentes formas de injetar serviços aqui e aqui.

- Compile a solução *(não é executar!)* e verifique que não tem nenhum erro.
- Para o formulário de encomenda, vamos utilizar ViewComponents, para tal, crie uma nova pasta "ViewComponents" dentro do projeto. Dentro desta pasta crie uma classe chamada "OrderPizzaViewComponent".
- A sua classe `OrderPizzaViewComponent` deve estender de `ViewComponent`. Injete também nesta classe o serviço OrderService seguindo o padrão já descrito: criar um campo privado de leitura, receber um valor por parâmetro no construtor, e atribuir ao seu campo o valor recebido.
- Crie um novo método "`InvokeAsync`" público e assíncrono que retorna `Task<IViewComponentResult>` e recebe como parâmetro um objeto do tipo `OrderResult?`.
- Implemente neste novo método, a lógica para que quando o objeto recebido seja null, retorne uma View "Order". Esta view já se encontra na solução na pasta "Views/Shared/Components/OrderPizza". Caso o objeto não seja null, retorne apenas uma View (sem passar argumentos) por agora.

Programação Visual
Trabalho de Laboratório nº 8

Nível 4

- Como pode verificar na view mencionada no nível anterior, esta precisa de preencher alguns elementos `select`. Para tal, acrescente no seu método `InvokeAsync` o código necessário para acrescentar ao ViewBag as SelectList necessárias.

Nota: Para os Ingredients terá de utilizar o serviço injetado, e para as Base e Size das pizzas deverá utilizar os enums.

- Na view Index dentro da pasta "Views/Home", invoque a ViewComponent através do seguinte código:

```
@await Component.InvokeAsync("OrderPizza")
```

- Execute a aplicação e verifique que se encontra no ecrã de encomenda de pizza e os campos do formulário estão corretamente preenchidos.
- No controlador OrderPizzaController, já incluído na solução, injete o OrderService, seguindo o processo que já fez anteriormente.
- Adicione uma nova ação Order, que recebe por parâmetro um objeto do tipo Pizza. Acrescente o attribute `HttpPost` a esta nova ação. Nesta ação, chame o método OrderPizza do OrderService, e use esse resultado como modelo da ViewComponent OrderPizza.

Nota: Para retornar uma ViewComponent em vez de uma View normal, basta fazer `return ViewComponent("NomeDaViewComponent", modelo);`

Programação Visual
Trabalho de Laboratório nº 8

Nível 5

- Crie uma nova view vazia "Result" dentro da pasta "Views/Shared/Components/OrderPizza".
- Nesta view injete o IngredientService através do seguinte código:

```
@using IPSPizza.Services.IngredientService;

@inject IIngredientService _ingredientService
```

- Adicione à view o modelo OrderResult
- Contrua um ecrã onde mostre o resumo da encomenda feita, deve incluir a Base, o tamanho, uma lista de ingredientes e o preço total da Pizza.

Nota: O seu modelo contém a pizza encomendada, no entanto esta apenas guarda os Ids dos ingredientes. Para poder obter o seu nome irá ter de usar o serviço injetado na view e pesquisar por Id na lista de ingredientes.

- Por fim, volte à classe "OrderPizzaViewComponent" e onde anteriorment retornava uma View sem parametros, passe a retorner a view "Result" passando como modelo o objeto que recebe como parâmetro.
- Corra a aplicação, encomende uma pizza e verifique que o seu ecrã de resultado contém os dados correto.

Desafio

- No OrderService, onde está a calcular o custo dos ingredientes, torne este código mais otimizado tirando partido do paralelismo. Pode calcular o valor de cada ingrediente em simultâneo, somando todos estes no fim.

Dicas: Use `Parallel.ForEach()` de forma a percorrer os ingredientes, e utilize `ConcurrentBag<decimal>` de forma a guardar os preços de cada ingrediente. O uso de ConcurrentBag permite que se mantenha *thread safety*.

Notas

Para os identificadores siga as convenções adotadas pelo C#, nomeadamente:

- A notação camelCase para o nome das variáveis locais e identificadores privados.
- A notação PascalCase para os nomes públicos dos métodos e classes
- Não utilize o símbolo '_' nos identificadores
- Não use abreviaturas